Homework 1: Classification With Naive Bayes

Sriharsha M S - sm39@illinois.edu | Vedprakash Mishra - vrm2@illinois.edu

Part 1 Accuracies (10 points)

Setup	Cross-validation Accuracy
Unprocessed data	0.7724550898203593
0-value elements ignored	0.7425149700598802

Part 1 Code Snippets (30 points)

1. Calculation of distribution parameters

```
def fit(self,features, labels):
    self.min_std = 0.00000001
    self.ntargets= np.unique(labels).shape[0]
    self.target_labels = np.unique(labels)
    self.nfeatures = features.shape[1]
    self.means = np.zeros((self.ntargets,self.nfeatures))
    self.stds = np.zeros((self.ntargets,self.nfeatures))
    self.priors = np.zeros(self.ntargets)
    for _index in range(self.ntargets):
        where_label = [label==self.target_labels[_index] for label in labels]
        self.means[_index] = np.nanmean(features[where_label],axis=0)
        self.stds[_index] =
np.clip(np.nanstd(features[where_label],axis=0),self.min_std,None)
        self.priors[_index] = np.log(np.sum(where_label)/len(labels))
```

2. Calculation of naive Bayes predictions

```
def predict(self,test_features):
    test_samples = test_features.shape[0]
    posterior = np.zeros((test_samples,self.ntargets))
    for target_label in range(self.ntargets):
        posterior[:,target_label] = self.priors[target_label] +
np.nansum(np.log(norm.pdf(test_features,self.means[target_label],self.stds[target_l
abel])),axis=1)
    label = self.target_labels[np.argmax(posterior, axis=1)]
    return label
```

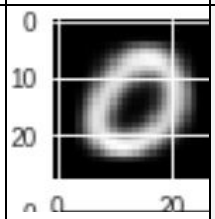
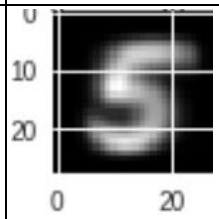
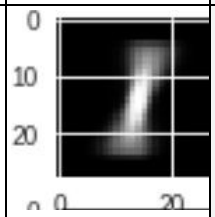
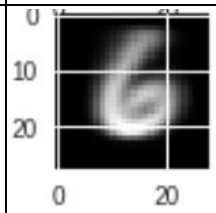
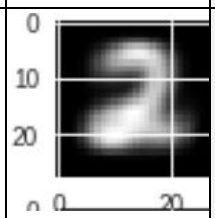
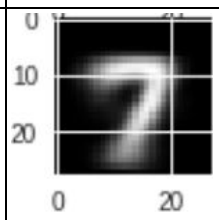
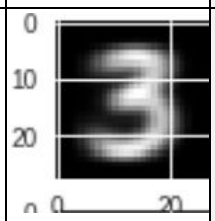
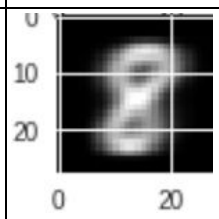
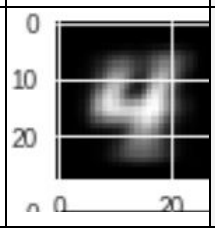
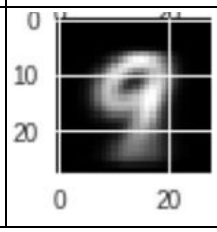
3. Test-train split code

```
def train_test_split(features, labels, test_size=0.2):
    np.random.seed(4)
    id = np.random.rand(len(features))>test_size
    features_train = features[id]
    labels_train = labels[id]
    features_test = features[np.invert(id)]
    labels_test = labels[np.invert(id)]
    return features_train, labels_train, features_test, labels_test
```

Part 2 MNIST Accuracies (20 points)

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	0.54625	0.5415
2	Gaussian + stretched	0.813033333333333	0.8234
3	Bernoulli + untouched	0.83645	0.8445
4	Bernoulli + stretched	0.795	0.8106
5	10 trees + 4 depth + untouched	0.71845	0.726
6	10 trees + 4 depth + stretched	0.744466666666666	0.7663
7	10 trees + 16 depth + untouched	0.99265	0.9486
8	10 trees + 16 depth + stretched	0.995833333333333	0.9549
9	30 trees + 4 depth + untouched	0.76895	0.7792
10	30 trees + 4 depth + stretched	0.768166666666666	0.7867
11	30 trees + 16 depth + untouched	0.996183333333333	0.9636
12	30 trees + 16 depth + stretched	0.9974	0.9646

Part 2A Digit Images (10 points)

Digit	Mean Image		Digit	Mean Image
0			5	
1			6	
2			7	
3			8	
4			9	

Part 2 Code (30 points)

- Calculation of the Normal distribution parameters

```
def fit(self,features, labels):
      self.min_std = 0.005
      self.ntargets= np.unique(labels).shape[0]
      self.target_labels = np.unique(labels)
      self.nfeatures = features.shape[1]
      self.means = np.zeros((self.ntargets,self.nfeatures))
      self.stds = np.zeros((self.ntargets,self.nfeatures))
      self.priors = np.zeros(self.ntargets)
      for _index in range(self.ntargets):
          where_label = [label==self.target_labels[_index] for label in labels]
          self.means[_index] = np.nanmean(features[where_label],axis=0)
          self.stds[_index] = np.clip(np.nanstd(features[where_label],axis=0),self.min_std,None)
          self.priors[_index] = np.log(np.sum(where_label)/len(labels))
```

- Calculation of the Bernoulli distribution parameters

```
def fit(self,features, labels):
        self.num_of_class = np.unique(labels).shape[0]
        self.target_labels = np.unique(labels)
        self.num_of_feature = features.shape[1]
        self.pblacks = np.zeros((self.num_of_class,self.num_of_feature))
        self.priors = np.zeros(self.num_of_class)
        for i in range(self.num_of_class):
            where_label = [l==self.target_labels[i] for l in labels]
            self.pblacks[i] = np.mean(features[where_label], axis=0)/255
            self.priors[i] = np.log(np.sum(where_label)/len(labels))
```

- Calculation of the Naive Bayes predictions

```
def predict(self,test_features):
        test_samples = test_features.shape[0]
        posterior = np.zeros((test_samples,self.ntargets))
        for target_label in range(self.ntargets):
            posterior[:,target_label] = self.priors[target_label] +
np.nansum(np.log(norm.pdf(test_features,self.means[target_label],self.stds[target_label])),axis
=1)
        label = self.target_labels[np.argmax(posterior, axis=1)]
        return label
```

- Training & Calculation of a decision tree predictions

```
def calculate_accuracy(stretched_bb_flag, model_name, use_sklearn=False, forest_params=(10,4)):
   ….
    if use_sklearn == False:
        if (model_name =='Normal'): classifier = GaussianNaiveBayes()
        elif (model_name =='Bernoulli'): classifier = BernoulliNaiveBayes()
    else:
        if (model_name =='Normal'): classifier = naive_bayes.GaussianNB()
        elif (model_name =='Bernoulli'): classifier = naive_bayes.BernoulliNB()
        elif (model_name =='decision_forest'): classifier =
RandomForestClassifier(n_estimators=forest_params[0], max_depth= forest_params[1],
random_state=4)
    classifier.fit(m_features_train, labels_train)
    return classifier.score(m_features_train, labels_train), classifier.score(m_features_test,
labels_test)
```

Homework 1: Classification With Naive Bayes

Problem 1: Diabetes Classification

Points: 40

A famous collection of data on whether a patient has diabetes, known as the Pima Indians dataset, and originally owned by the National Institute of Diabetes and Digestive and Kidney Diseases can be found at Kaggle. Download this dataset from https://www.kaggle.com/kumargh/pimaindiansdiabetescsv. This data has a set of attributes of patients, and a categorical variable telling whether the patient is diabetic or not. For several attributes in this data set, a value of 0 may indicate a missing value of the variable. There are a total of 767 data-points.

Part 1A

Build a simple naive Bayes classifier to classify this data set. You should use a normal distribution to model each of the class-conditional distributions.

Compute an estimate of the accuracy of the classifier by averaging over 10 test-train splits. Each split should randomly assign 20% of the data to test, and the rest to train.
You should write this classifier and the test-train split code yourself (it's quite straight-forward). Libraries can be used to load & hold the data.

Answer Part 1A

To build a somple naive Bayes classifier to classify Pima Indians dataset. We will be using Python 3 in Google Colab.

Set up

Load required libraries

```
import pandas as pd
import numpy as np
import math
from scipy.stats import norm

import pandas_profiling

import pickle

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

Load dataset

Access https://www.kaggle.com/kumargh/pimaindiansdiabetescsv, download the dataset, it will download pimaindiansdiabetescsv.zip.

Dataset dictionary

Following are the datset details as per kaggle.

About this file
This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within ve years.

Fields description follow:

preg = Number of times pregnant

plas = Plasma glucose concentration a 2 hours in an oral glucose tolerance test

pres = Diastolic blood pressure (mm Hg)

skin = Triceps skin fold thickness (mm)

test = 2-Hour serum insulin (mu U/ml)

mass = Body mass index (weight in kg/(height in m)^2)

pedi = Diabetes pedigree function

age = Age (years)

class = Class variable (1:tested positive for diabetes, 0: tested negative for diabetes)

Columns, first row is sample data, ignoring to obtain dataset of rows 767, as mentioned in home work assignment link.

6 Pregnancies

148 Glucose

72 BloodPressure

35 SkinThickness

0 Insulin

33.6 BMI

0.627 DiabetesPedigreeFunction

50 Age

1 Class

To access the dataset in Google Colab you can either use Github or Google Drive. We will be accessing dataset via Google Drive. Unzip the pimaindiansdiabetescsv.zip, add pima-indians-diabetes.csv to a known folder in Google Drive, this folder path in drive will be accessed later to load dataset.

We added the pima-indians-diabetes.csv to Google Drive folder /My Drive/UISC-MCS-DS/CS498AML/homework_1/1a/data/.

- Mount Google Drive to access data Note: This is not required if you are not using Google colab

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=Tr
ue).
```

Load pima-indians-diabetes.csv Dataset and save it as a pickle object

```
pima_indias_diabetes_data = pd.read_csv("/content/gdrive/My Drive/UIUC-MCS-DS/CS498AML/homework_1/1a/data/pima-indians-diabete
s.csv")
pickle.dump(pima_indias_diabetes_data, open( '/content/gdrive/My Drive/UIUC-MCS-DS/CS498AML/homework_1/1a/data/pima_indias_dia
betes_data.pkl','wb'))
pima_indias_diabetes_data.head()
```

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

Exploratory Data Analysis

Validate dataset

```
# rename column names
pima_indias_diabetes_data.columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPe
digreeFunction', 'Age', 'Class']
pima_indias_diabetes_data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Class
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

```
# count number of dataset
print("There are a total of", len(pima_indias_diabetes_data),"data-points")
```

```
There are a total of 767 data-points
```

```
pima_indias_diabetes_data_features = pima_indias_diabetes_data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'I
nsulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
pima_indias_diabetes_data_labels = pima_indias_diabetes_data[['Class']]
print(pima_indias_diabetes_data_features.shape)
print(pima_indias_diabetes_data_labels.shape)
```

```
(767, 8)
(767, 1)
```

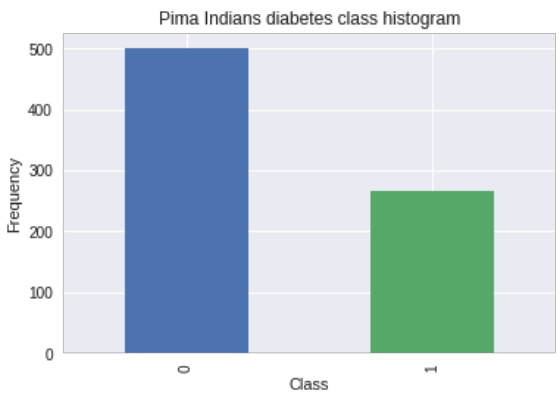
There are 767 observations of 8 different features.

Analyse label

```
count_classes = pd.value_counts(pima_indias_diabetes_data['Class'], sort = True).sort_index()
count_classes.plot(kind = 'bar')
plt.title("Pima Indians diabetes class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
```

```
pima_indias_diabetes_data.groupby('Class')['Class'].count()
```

```
Class
0    500
1    267
Name: Class, dtype: int64
```



png

There are 500 Pima Indians 0: tested negative for diabetes, 267 :tested positive for diabetes.

Data distribution analysis for each feature and class label

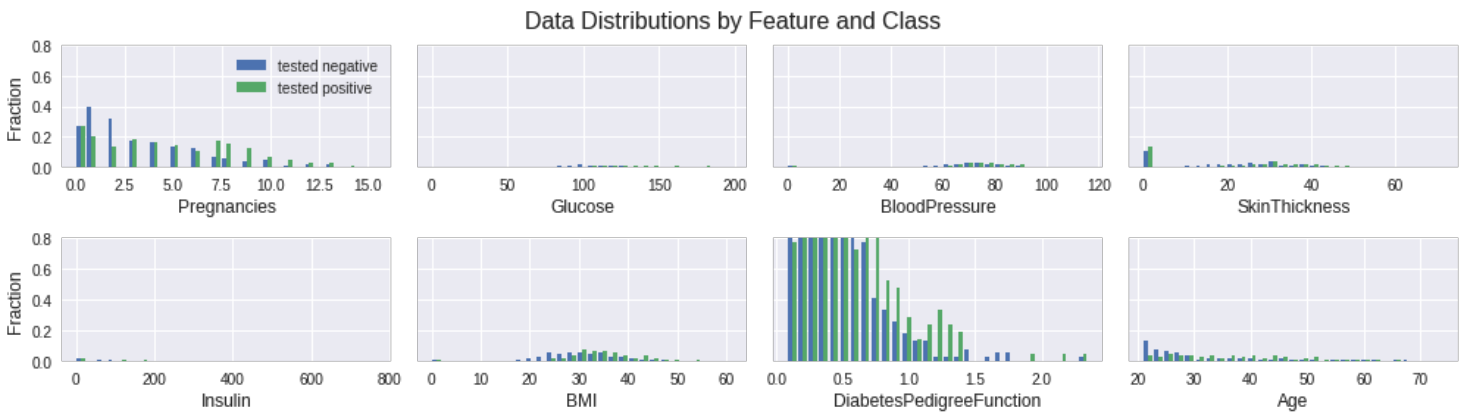
Plot the data by each feature

```
axarr = [[]]*len(pima_indias_diabetes_data_features.columns)
columns = 4
rows = int( np.ceil( len(pima_indias_diabetes_data_features.columns) / columns ) )
f, fig = plt.subplots( figsize=(columns*3.5, rows*2) )

f.suptitle('Data Distributions by Feature and Class', size=16)

for i, col in enumerate(pima_indias_diabetes_data_features.columns[:]):
    axarr[i] = plt.subplot2grid( (int(rows), int(columns)), (int(i//columns), int(i%columns)) )
    axarr[i].hist( [ pima_indias_diabetes_data.loc[ pima_indias_diabetes_data.Class == 0, col ], pima_indias_diabetes_data.loc
[ pima_indias_diabetes_data.Class == 1, col ] ], label=['tested negative','tested positive'],
                           bins=np.linspace( np.percentile(pima_indias_diabetes_data[col],0.1), np.percentile(pima_indias_diabe
tes_data[col],99.9), 30 ),
                           normed=True )
    axarr[i].set_xlabel(col, size=12)
    axarr[i].set_ylim([0,0.8])
    axarr[i].tick_params(axis='both', labelsize=10)
    if i == 0:
        legend = axarr[i].legend()
        legend.get_frame().set_facecolor('white')
    if i%4 != 0 :
        axarr[i].tick_params(axis='y', left='off', labelleft='off')
    else:
        axarr[i].set_ylabel('Fraction',size=12)

plt.tight_layout(rect=[0,0,1,0.95]) # xmin, ymin, xmax, ymax
plt.show()
```



png

Classify Dataset - Build a simple naive Bayes classifier

Split data

```
def train_test_split(features, labels, test_size=0.2):
    np.random.seed(4)
    id = np.random.rand(len(features))>test_size
    #print(id)
    features_train = features[id]
    labels_train = labels[id]
    features_test = features[np.invert(id)]
    labels_test = labels[np.invert(id)]
    return features_train, labels_train, features_test, labels_test


features = pima_indias_diabetes_data.drop('Class', axis = 1)
labels = pima_indias_diabetes_data['Class']
```

Gaussian Naive Bayes classifier

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution.

The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma {y}^{2} }} exp \left (-\frac{(x_i-\mu {y})^2}{2\sigma _{y}^{2}} \right) $

we will use norm.pdf to calculate probablity density function.

```
class GaussianNaiveBayes():
    """The Gaussian Naive Bayes classifier. """
    def fit(self,features, labels):
      """
      for each label and feature combination we need to calculate the std and mean value from the features & labels.
      """
      self.min_std = 0.00000001
      self.ntargets= np.unique(labels).shape[0]
      self.target_labels = np.unique(labels)
      self.nfeatures = features.shape[1]
      self.means = np.zeros((self.ntargets,self.nfeatures))
      self.stds = np.zeros((self.ntargets,self.nfeatures))
      self.priors = np.zeros(self.ntargets)
      for _index in range(self.ntargets):
          # Get the boolean vector to filter for labels = i
          where_label = [label==self.target_labels[_index] for label in labels]
```

```
        self.means[_index] = np.nanmean(features[where_label],axis=0)
        # To avoid devide by 0/very small value issue, add a min for standard deviation to min_std = 0.00000001
        self.stds[_index] = np.clip(np.nanstd(features[where_label],axis=0),self.min_std,None)
        #print(self.means[_index], self.stds[_index])
        #Calculate the prior for given label
        self.priors[_index] = np.log(np.sum(where_label)/len(labels))

    def predict(self,test_features):
        """ Classification using Bayes Rule P(Y|X) = P(X|Y)*P(Y)/P(X),
          or Posterior = Likelihood * Prior / Scaling Factor
          P(Y|X) - The posterior is the probability that sample x is of class y given the
                   feature values of x being distributed according to distribution of y and the prior.
          P(X|Y) - Likelihood of data X given class distribution Y.
                   Gaussian distribution (given by _calculate_likelihood)
          P(Y)   - Prior (given by _calculate_prior)
          P(X)   - Scales the posterior to make it a proper probability distribution.
                   This term is ignored in this implementation since it doesn't affect
                   which class distribution the sample is most likely to belong to.
      Classifies the sample as the class that results in the largest P(Y|X) (posterior)
      """
        test_samples = test_features.shape[0]
        posterior = np.zeros((test_samples,self.ntargets))
        # Naive assumption (independence):
        # P(x1,x2,x3|Y) = P(x1|Y)*P(x2|Y)*P(x3|Y)
        # Posterior is product of prior and likelihoods (ignoring scaling factor)
        for target_label in range(self.ntargets):
            posterior[:,target_label] = self.priors[target_label] + np.nansum(np.log(norm.pdf(test_features,self.means[target_
label],self.stds[target_label])),axis=1)
        label = self.target_labels[np.argmax(posterior, axis=1)]
        return label


    def score(self,X_test, y_test):
        y_predict = self.predict(X_test)
        return (y_predict == y_test).mean()
```

Compute an estimate of the accuracy of the classifier by averaging over 10 test-train splits. Each split should randomly assign 20% of the data to test, and the rest to train.

```
print(features.shape)



test_accuracy_iterations =[]
# for 10 iterations
for i in range(10):
  features_train, labels_train, features_test, labels_test = train_test_split(features.values,
                                 labels.values, test_size=0.2)
  nb = GaussianNaiveBayes()

  nb.fit(features_train, labels_train)
  test_accuracy = nb.score(features_test, labels_test)
  test_accuracy_iterations.append(test_accuracy)

print(test_accuracy_iterations)
print("Average test accuracy", np.mean(test_accuracy_iterations))
```

```
(767, 8)
[0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.772
4550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593]
Average test accuracy 0.7724550898203593
```

Validate using scikit learn naive bayes

```
from sklearn import naive_bayes

test_accuracy_iterations =[]
# for 10 iterations
for i in range(10):
  features_train, labels_train, features_test, labels_test = train_test_split(features.values, labels.values, test_size=0.2)
```

```
  snb = naive_bayes.GaussianNB()

  snb.fit(features_train, labels_train)
  test_accuracy = snb.score(features_test, labels_test)
  test_accuracy_iterations.append(test_accuracy)

print(test_accuracy_iterations)
print("Average test accuracy", np.mean(test_accuracy_iterations))
```

```
[0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593, 0.772
4550898203593, 0.7724550898203593, 0.7724550898203593, 0.7724550898203593]
Average test accuracy 0.7724550898203593
```

Part 1B

Now adjust your code so that, for attribute 3 (Diastolic blood pressure), attribute 4 (Triceps skinfold thickness), attribute 6 (Body mass index), and attribute 8 (Age), it regards a value of 0 as a missing value when estimating the class-conditional distributions, and the posterior.

- Compute an estimate of the accuracy of the classifier by averaging over 10 test-train splits.

Answer Part 1B

All the above work done for Part 1A will be reused for Part 1B.

We will mainly be processing data to remove missing values which are 0.

Impute missing values 0 for attributes 3 (Diastolic blood pressure), attribute 4 (Triceps skinfold thickness), attribute 6 (Body mass index), and attribute 8 (Age) as np.NaN.

Check how many missing values are present.

```
pima_indias_diabetes_data[['BloodPressure','SkinThickness','BMI','Age']]= pima_indias_diabetes_data[['BloodPressure','SkinThic
kness','BMI','Age']].replace(0, np.NaN)

print(pima_indias_diabetes_data.isnull().sum())
```

```
Pregnancies                   0
Glucose                       0
BloodPressure                35
SkinThickness               227
Insulin                       0
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Class                         0
dtype: int64
```

```
#pima_indias_diabetes_data.dropna(inplace=True)

print(pima_indias_diabetes_data.isnull().sum())
```

```
Pregnancies                   0
Glucose                       0
BloodPressure                35
SkinThickness               227
Insulin                       0
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Class                         0
dtype: int64
```

```
pima_indias_diabetes_data.shape
```

```
(767, 9)
```

After processing the missing data split data and build new model with new train dataset and test he accuracy.

```
processed_features = pima_indias_diabetes_data.drop('Class', axis = 1)
processed_labels = pima_indias_diabetes_data['Class']

#scaler = StandardScaler()
#normal_processed_features = scaler.fit_transform(processed_features)
```

```
processed_test_accuracy_iterations =[]
# for 10 iterations
for i in range(10):
  p_features_train, p_labels_train, p_features_test, p_labels_test = train_test_split(processed_features.values, processed_lab
els.values, test_size=0.2)
  p_nb = GaussianNaiveBayes()

  p_nb.fit(p_features_train, p_labels_train)
  p_test_accuracy = p_nb.score(p_features_test, p_labels_test)
  processed_test_accuracy_iterations.append(p_test_accuracy)

print(processed_test_accuracy_iterations)
print("Average test accuracy for processed data", np.mean(processed_test_accuracy_iterations))
```

```
[0.7425149700598802, 0.7425149700598802, 0.7425149700598802, 0.7425149700598802, 0.7425149700598802, 0.7425149700598802, 0.742
5149700598802, 0.7425149700598802, 0.7425149700598802, 0.7425149700598802]
Average test accuracy for processed data 0.7425149700598802
```

References

Following are various resources referred while writing this solution

- Github projects https://github.com/sriharshams/mlnd/
- Code of codyznash https://github.com/codyznash/GANs_for_Credit_Card_Data
- Tutorials of https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
- Naive Bayes in Scikit-Learn: Implementation of naive bayes in the scikit-learn library.
- ML from scratch
- Naive Bayes documentation: Scikit-Learn documentation and sample code for Naive Bayes
- Naive Bayes Classifiers https://www.geeksforgeeks.org/naive-bayes-classifiers/
- Naive Bayes Classifier From Scratch
- https://chrisalbon.com/machine_learning/naive_bayes/naive_bayes_classifier_from_scratch/
- Naive Bayes from scratch in python http://kenzotakahashi.github.io/naive-bayes-from-scratch-in-python.html
- Applied Machine Learning, D.A. Forsyth, (approximate 18'th draft)
- Piazza & Slack discussions on CS–498 Spring 2019

Homework 1: Question 2

Problem 2: MNIST Image Classification

Points: 60

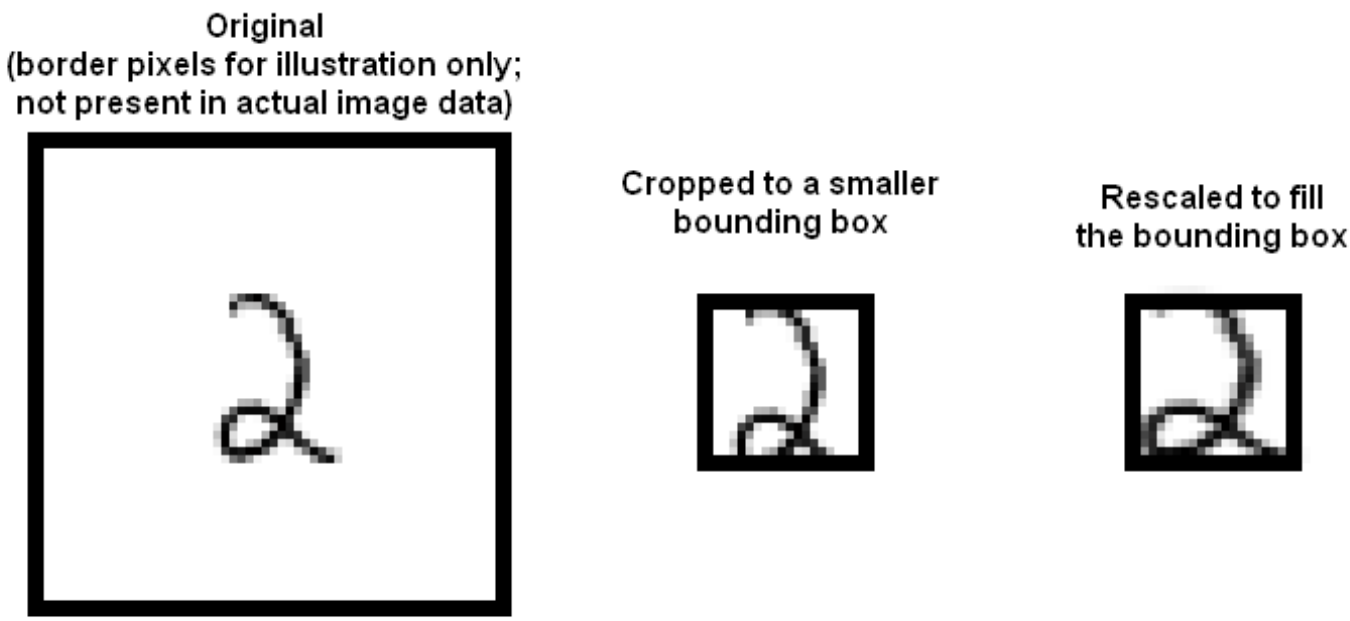
The MNIST dataset is a dataset of 60,000 training and 10,000 test examples of handwritten digits, originally constructed by Yann Lecun, Corinna Cortes, and Christopher J.C. Burges. It is very widely used to check simple methods. There are 10 classes in total ("0" to "9"). This dataset has been extensively studied, and there is a history of methods and feature constructions at https://en.wikipedia.org/wiki/MNIST_database and at the original site, http://yann.lecun.com/exdb/mnist/. You should notice that the best methods perform extremely well.

(Updated 1/19) The http://yann.lecun.com/exdb/mnist/ dataset is stored in an unusual format, described in detail on the page. You do not have to write your own reader. A web search should yield solutions for both Python and R. For Python, https://pypi.org/project/python-mnist/ should work. For R, there is reader code available at https://stackoverflow.com/questions/21521571/how-to-read-mnist-database-in-r. Please note that if you follow the recommendations in the accepted answer there at https://stackoverflow.com/a/21524980, you must also provide the readBin call with the flag signed=FALSE since the data values are stored as unsigned integers.

The dataset consists of 28 x 28 images. These were originally binary images, but appear to be grey level images as a result of some anti-aliasing. I will ignore mid-grey pixels (there aren't many of them) and call dark pixels "ink pixels", and light pixels "paper pixels"; you can modify the data values with a threshold to specify the distinction, as described here https://en.wikipedia.org/wiki/Thresholding_(image_processing). The digit has been centered in the image by centering the center of gravity of the image pixels, but as mentioned on the original site, this is probably not ideal. Here are some options for re-centering the digits that I will refer to in the exercises.

Untouched: Do not re-center the digits, but use the images as is.
Bounding box: Construct a 20 x 20 bounding box so that the horizontal (resp. vertical) range of ink pixels is centered in the box.
Stretched bounding box: Construct a 20 x 20 bounding box so that the horizontal (resp. vertical) range of ink pixels runs the full horizontal (resp. vertical) range of the box. Obtaining this representation will involve rescaling image pixels: you find the horizontal and vertical ink range, cut that out of the original image, then resize the result to 20 x 20. Once the image has been re-centered, you can compute

features.
Here are some pictures, which may help
bounding box



mnist_image

Part 2A: MNIST using naive Bayes

Model each class of the dataset using a Normal distribution and (separately) a Bernoulli distribution for both untouched images v. stretched bounding boxes, using 20 x 20 for your bounding box dimension. This should result in 4 total models. Use the training set to calculate the distribution parameters.

You must write the naive Bayes prediction code. The distribution parameters can be calculated manually or via libraries. Additionally, we recommend using a library to load the MNIST data (e.g. python-mnist or scikit-learn) and to rescale the images (e.g. openCV).

Compute the accuracy values for the four combinations of Normal v. Bernoulli distributions for both untouched images v. stretched bounding boxes. Both the training and test set accuracy will be reported.
For each digit, plot the mean pixel values calculated for the Normal distribution of the untouched images. In Python, a library such as matplotlib should prove useful.

Answer Part 2A:

To build a somple naive Bayes classifier to classify MNIST data dataset. We will be using Python 3 in Google Colab.

Set up

Load required libraries

```
!pip install --upgrade --force-reinstall python-mnist
```

```
Collecting python-mnist
  Downloading https://files.pythonhosted.org/packages/05/9c/f1c1e4d011b01ac436bba0ac67
15b3f988bb7f8fec6f21f89cf820aa33e1/python-mnist-0.6.tar.gz
Building wheels for collected packages: python-mnist
  Running setup.py bdist_wheel for python-mnist ... [?25ldone
[?25h  Stored in directory: /root/.cache/pip/wheels/28/29/36/408f83545a511c43d03ef997a
1dc99b49ccd5f9f306ed92468
Successfully built python-mnist
Installing collected packages: python-mnist
Successfully installed python-mnist-0.6
```

```
import pandas as pd
import numpy as np
from scipy.stats import norm
from scipy.stats import bernoulli
from mnist import MNIST
import matplotlib.pyplot as plt
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn import naive_bayes
import warnings
warnings.filterwarnings("ignore")
```

Load dataset

Access http://yann.lecun.com/exdb/mnist/, download the dataset.

There are 4 files:

train-images-idx3-ubyte: training set images

train-labels-idx1-ubyte: training set labels

t10k-images-idx3-ubyte: test set images

t10k-labels-idx1-ubyte: test set labels

The training set contains 60000 examples, and the test set 10000 examples.

Dataset dictionary

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset][type] [value][description]

0000 32 bit integer 0x00000801(2049) magic number (MSB first)

0004 32 bit integer 60000 number of items

0008 unsigned byte ?? label

0009 unsigned byte ?? label

……..

xxxx unsigned byte ?? label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset][type] [value][description]

0000 32 bit integer 0x00000803(2051) magic number

0004 32 bit integer 60000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte ?? pixel

0017 unsigned byte ?? pixel

……..

xxxx unsigned byte ?? pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

[offset][type] [value][description]

0000 32 bit integer 0x00000801(2049) magic number (MSB first)

0004 32 bit integer 10000 number of items

0008 unsigned byte ?? label

0009 unsigned byte ?? label

……..

xxxx unsigned byte ?? label

The labels values are 0 to 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

[offset][type] [value][description]

0000 32 bit integer 0x00000803(2051) magic number

0004 32 bit integer 10000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte ?? pixel

0017 unsigned byte ?? pixel

……..

xxxx unsigned byte ?? pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

To access the dataset in Google Colab you can either use Github or Google Drive. We will be accessing dataset via Google Drive. Unzip the pimaindiansdiabetescsv.zip, add pima-indians-diabetes.csv to a known folder in Google Drive, this folder path in drive will be accessed later to load dataset.

We added the pima-indians-diabetes.csv to Google Drive folder /My Drive/UISC-MCS-DS/CS498AML/homework_1/2a/data/.

- Mount Google Drive to access data Note: This is not required if you are not using Google colab

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Reading the dataset

Following instructions here https://pypi.org/project/python-mnist/, we were able to load dataset using MNIST library

```
mndata = MNIST('/content/gdrive/My Drive/UIUC-MCS-DS/CS498AML/homework_1/data')
features_train, labels_train = mndata.load_training()
features_test, labels_test = mndata.load_testing()
```

Data processing

Bounding box: Construct a 20 x 20 bounding box so that the horizontal (resp. vertical) range of ink pixels is centered in the box.

Stretched bounding box: Construct a 20 x 20 bounding box so that the horizontal (resp. vertical) range of ink pixels runs the full horizontal (resp. vertical) range of the box. Obtaining this representation will involve rescaling image pixels: you find the horizontal and vertical ink range, cut that out of the original image, then resize the result to 20 x 20. Once the image has been re-centered, you can compute features.

```
class data_preprocessor():
    def __init__(self, image_size=28, thresholding=50, stretched_bb_flag=False, resize
=20):
        self.image_size = image_size
        self.thresholding = thresholding
        self.stretched_bb_flag = stretched_bb_flag
        self.resize = resize




    def preprocess(self,features):
        self.count_features=features.shape[0]

        if self.stretched_bb_flag:
            features_reshape = features.reshape(self.count_features,self.image_size,se
lf.image_size).astype(float)
```

```
            features_reshape_stretched_boundingbox = np.zeros((self.count_features,sel
f.resize,self.resize))

            for i in range(self.count_features):
                features_reshape_stretched_boundingbox[i]=self.stretched_boundingbox_s
ingle(features_reshape[i])
            features_reshape_stretched_boundingbox=features_reshape_stretched_bounding
box.reshape(self.count_features,self.resize*self.resize)

            # Image processing, thresholding
            features_thresholding = (features_reshape_stretched_boundingbox >self.thre
sholding).astype(float)*255

            return features_thresholding

        # Image processing, thresholding, threshold at 50
        features_thresholding = (features >self.thresholding).astype(float)*255

        return features_thresholding

    def stretched_boundingbox_single(self, img):
        lower_bound =  np.where(np.any(img,axis=1))[0][0]
        upper_bound =  np.where(np.any(img,axis=1))[0][-1]
        left_bound =  np.where(np.any(img,axis=0))[0][0]
        right_bound =  np.where(np.any(img,axis=0))[0][-1]
        stretched_boundingbox = cv2.resize(img[lower_bound:upper_bound,left_bound:righ
t_bound], (self.resize, self.resize))
        return stretched_boundingbox
```

```
# Preprocessing option1: Original data (without stretched bounding box) with threshold
ing
img_processor = data_preprocessor(stretched_bb_flag=False)
features_train = img_processor.preprocess(np.array(features_train))
features_test = img_processor.preprocess(np.array(features_test))

# Preprocessing option2: Stretched Bounding Box with thresholding
img_processor2 = data_preprocessor(stretched_bb_flag=True)
features_train_stretched_boundingbox  = img_processor2.preprocess(np.array(features_tr
ain))
features_test_test_stretched_boundingbox  = img_processor2.preprocess(np.array(feature
s_test))

# Convert labels to array
labels_train = np.array(labels_train)
labels_test = np.array(labels_test)

print("training ", features_train.shape)
print("training ", len(labels_train))
```

```
training  (60000, 784)
training  60000
```

There are 60000 images for training.

Classify Dataset - Build a simple naive Bayes classifier

Gaussian Naive Bayes classifier

Reusing Gaussian Naive Bayes from Homework 1a

```
class GaussianNaiveBayes():
    """The Gaussian Naive Bayes classifier. """
    def fit(self,features, labels):
      """
      for each label and feature combination we need to calculate the std and mean val
ue from the features & labels.
      """
      self.min_std = 0.005
      self.ntargets= np.unique(labels).shape[0]
      self.target_labels = np.unique(labels)
      self.nfeatures = features.shape[1]
      self.means = np.zeros((self.ntargets,self.nfeatures))
      self.stds = np.zeros((self.ntargets,self.nfeatures))
      self.priors = np.zeros(self.ntargets)
      for _index in range(self.ntargets):
          # Get the boolean vector to filter for labels = i
          where_label = [label==self.target_labels[_index] for label in labels]
          self.means[_index] = np.nanmean(features[where_label],axis=0)
          # To avoid divide by 0/very small value issue, add a min for standard deviat
ion to min_std = 0.00000001
          self.stds[_index] = np.clip(np.nanstd(features[where_label],axis=0),self.min
_std,None)
          #print(self.means[_index], self.stds[_index])
          #Calculate the prior for given label
          self.priors[_index] = np.log(np.sum(where_label)/len(labels))

    def predict(self,test_features):
        """ Classification using Bayes Rule P(Y|X) = P(X|Y)*P(Y)/P(X),
          or Posterior = Likelihood * Prior / Scaling Factor
          P(Y|X) - The posterior is the probability that sample x is of class y given
the
                   feature values of x being distributed according to distribution of
y and the prior.
          P(X|Y) - Likelihood of data X given class distribution Y.
                   Gaussian distribution (given by norm.pdf)
          P(Y)   - Prior (given by _calculate_prior)
          P(X)   - Scales the posterior to make it a proper probability distribution.
```

```
                    This term is ignored in this implementation since it doesn't affect
                    which class distribution the sample is most likely to belong to.
         Classifies the sample as the class that results in the largest P(Y|X) (poster
ior)
         """
        test_samples = test_features.shape[0]
        posterior = np.zeros((test_samples,self.ntargets))
        # Naive assumption (independence):
        # P(x1,x2,x3|Y) = P(x1|Y)*P(x2|Y)*P(x3|Y)
        # Posterior is product of prior and likelihoods (ignoring scaling factor)
        for target_label in range(self.ntargets):
            posterior[:,target_label] = self.priors[target_label] + np.nansum(np.log(n
orm.pdf(test_features,self.means[target_label],self.stds[target_label])),axis=1)
        label = self.target_labels[np.argmax(posterior, axis=1)]
        return label


    def score(self, features_test, labels_test):
        labels_predict = self.predict(features_test)
        return (labels_predict == labels_test).mean()
```

Bernoulli Naive Bayes classifier

Bernoulli Naive Bayes classifier code

```
class BernoulliNaiveBayes():
    """The Bernoulli Naive Bayes classifier. """

    def fit(self,features, labels):
        """
        for each label and feature combination we need to calculate the std and mean v
alue from the features & labels.
        """
        self.num_of_class = np.unique(labels).shape[0]
        self.target_labels = np.unique(labels)
        self.num_of_feature = features.shape[1]
        self.pblacks = np.zeros((self.num_of_class,self.num_of_feature))
        self.priors = np.zeros(self.num_of_class)
        for i in range(self.num_of_class):
            # Get the boolean vector to filter for y = i
            where_label = [l==self.target_labels[i] for l in labels]
            self.pblacks[i] = np.mean(features[where_label], axis=0)/255
            # calculate priors
            self.priors[i] = np.log(np.sum(where_label)/len(labels))

    def predict(self,test_features):
        """ Classification using Bayes Rule P(Y|X) = P(X|Y)*P(Y)/P(X),
          or Posterior = Likelihood * Prior / Scaling Factor
          P(Y|X) - The posterior is the probability that sample x is of class y given
```

```
the
                    feature values of x being distributed according to distribution of
y and the prior.
          P(X|Y) - Likelihood of data X given class distribution Y.
                   Bernoulli distribution (given by bernoulli.pmf)
          P(Y)   - Prior (given by _calculate_prior)
          P(X)   - Scales the posterior to make it a proper probability distribution.
                   This term is ignored in this implementation since it doesn't affect
                   which class distribution the sample is most likely to belong to.
         Classifies the sample as the class that results in the largest P(Y|X) (poster
ior)
        """
        samples = test_features.shape[0]
        posterior = np.zeros((samples,self.num_of_class))
        for i in range(self.num_of_class):
            posterior[:,i] = self.priors[i] + np.sum(np.log(bernoulli.pmf(test_feature
s/255,self.pblacks[i])),axis=1)
        label = self.target_labels[np.argmax(posterior, axis=1)]
        return label

    def score(self, features_test, labels_test):
        labels_predict = self.predict(features_test)
        return (labels_predict == labels_test).mean()
```

Calculate Accuracy

```
# Use RandomForestClassifier to classify MNIST
from sklearn.ensemble import RandomForestClassifier

def calculate_accuracy(stretched_bb_flag, model_name, use_sklearn=False, forest_params
=(10,4)):
    if stretched_bb_flag:
        m_features_train = features_train_stretched_boundingbox
        m_features_test = features_test_test_stretched_boundingbox
    else:
        m_features_train = features_train
        m_features_test = features_test

    if use_sklearn == False:
        if (model_name =='Normal'):
            classifier = GaussianNaiveBayes()
        elif (model_name =='Bernoulli'):
            classifier = BernoulliNaiveBayes()
    else:
        if (model_name =='Normal'):
            classifier = naive_bayes.GaussianNB()
        elif (model_name =='Bernoulli'):
            classifier = naive_bayes.BernoulliNB()
        # Part 2B: MNIST using Decision Forest
```

```
        elif (model_name =='decision_forest'):
            classifier = RandomForestClassifier(n_estimators=forest_params[0], max_dep
th= forest_params[1], random_state=4)

    classifier.fit(m_features_train, labels_train)


    return classifier.score(m_features_train, labels_train), classifier.score(m_featur
es_test, labels_test)
```

Compute the accuracy values for the four combinations of Normal v. Bernoulli distributions for both untouched images v. stretched bounding boxes. Both the training and test set accuracy will be reported.

Classify MNIST using a decision forest.
For your forest construction, you should investigate four cases. Your cases are: number of trees = (10, 30) X maximum depth = (4, 16). You should compute your accuracy for each of the following cases: untouched raw pixels; stretched bounding box. This yields a total of 8 slightly different classifiers. Please use 20 x 20 for your bounding box dimensions.

```
print("Gaussian + untouched: train & test accuracy- ", calculate_accuracy(False, 'Norm
al'))

print("Gaussian + stretched: train & test accuracy- ", calculate_accuracy(True,  'Norm
al'))

print("Bernoulli + untouched: train & test accuracy- ", calculate_accuracy(False, 'Ber
noulli'))

print("Bernoulli + stretched: train & test accuracy- ", calculate_accuracy(True, 'Bern
oulli'))
```

```
Gaussian + untouched: train & test accuracy-  (0.54625, 0.5415)
Gaussian + stretched: train & test accuracy-  (0.8130333333333334, 0.8234)
Bernoulli + untouched: train & test accuracy-  (0.83645, 0.8445)
Bernoulli + stretched: train & test accuracy-  (0.795, 0.8106)
```

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	0.54625	0.5415
2	Gaussian + stretched	0.8130333333333334	0.8234
3	Bernoulli + untouched	0.83645	0.8445
4	Bernoulli + stretched	0.795	0.8106

Validate using scikit learn library

```
print("Gaussian + untouched: train & test accuracy- ", calculate_accuracy(False, 'Norm
al', True))

print("Gaussian + stretched: train & test accuracy- ", calculate_accuracy(True, 'Norma
l', use_sklearn=True))

print("Bernoulli + untouche: train & test accuracy- ", calculate_accuracy(False, 'Bern
oulli', use_sklearn=True))

print("Bernoulli + stretched: train & test accuracy- ", calculate_accuracy(True, 'Bern
oulli', use_sklearn=True))
```

```
Gaussian + untouched: train & test accuracy-  (0.5455833333333333, 0.5398)
Gaussian + stretched: train & test accuracy-  (0.8130333333333334, 0.8234)
Bernoulli + untouche: train & test accuracy-  (0.8343666666666667, 0.8445)
Bernoulli + stretched: train & test accuracy-  (0.7950166666666667, 0.8107)
```

Part 2A Digit Images

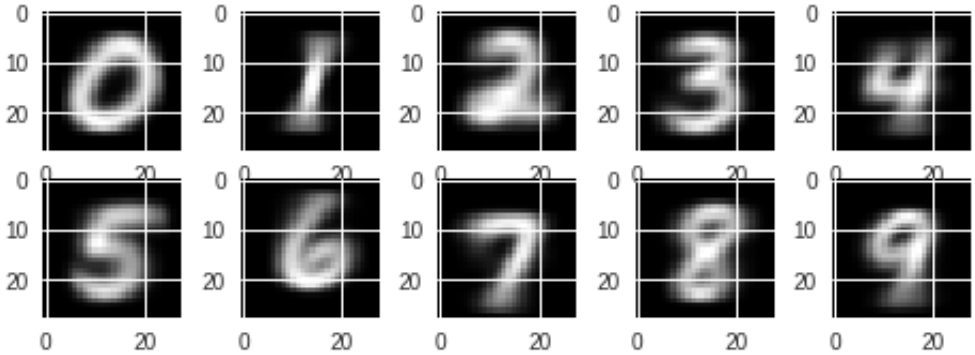
For each digit, plot the mean pixel values calculated for the Normal distribution of the untouched images. In Python, a library such as matplotlib should prove useful.

```
from pylab import imshow, show, cm
# Visualization Guassian Naive Bayes trained (on original data with thresholding) clas
sifier with mnist datase
nb = GaussianNaiveBayes()
nb.fit(features_train,labels_train)

def view_image(image, label=""):
    """View a single image."""
    #print("Label: %s" % label)
    imshow(image, cmap=cm.gray)
    show()

print("Normal distribution of untouched images")
for i in range(10):
    plt.subplot(4, 5, i+1)
    #view_image(nb.means[i].reshape(28,28), i)
    imshow(nb.means[i].reshape(28,28),cmap='gray')
```

```
Normal distribution of untouched images
```

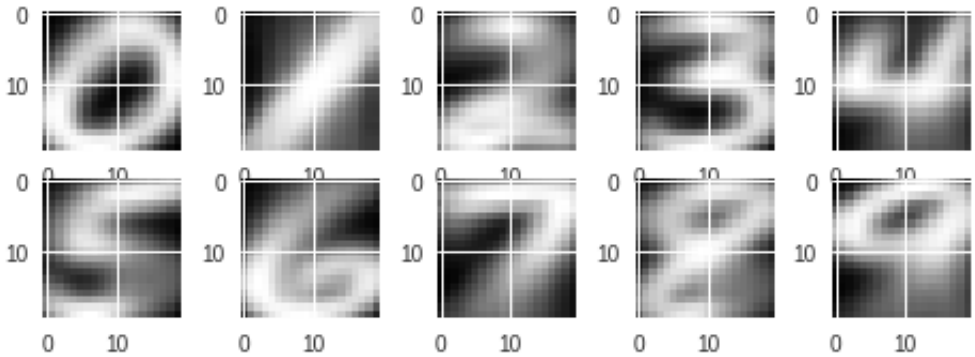


png

```
nb_stretched = GaussianNaiveBayes()
nb_stretched.fit(features_train_stretched_boundingbox, labels_train)

print("Normal distribution of stretched images")
for i in range(10):
    plt.subplot(4, 5, i+1)
    imshow(nb_stretched.means[i].reshape(20,20),cmap='gray')
```

```
Normal distribution of stretched images
```

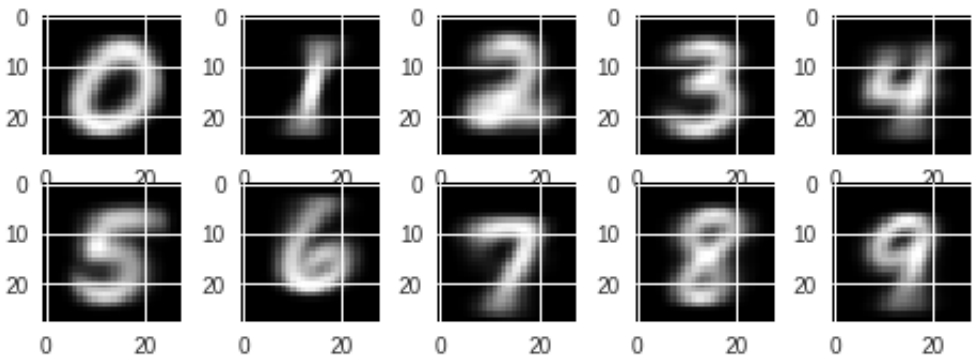


png

```
bernoulli_nb = BernoulliNaiveBayes()
bernoulli_nb.fit(features_train,labels_train)

print("Bernoulli distribution of untouched images")
for i in range(10):
    plt.subplot(4, 5, i+11)
    #view_image(np.array(bernoulli_nb.pblacks[i]).reshape(28,28), i)
    imshow(np.array(bernoulli_nb.pblacks[i]).reshape(28,28),cmap='gray')
```

```
Bernoulli distribution of untouched images
```

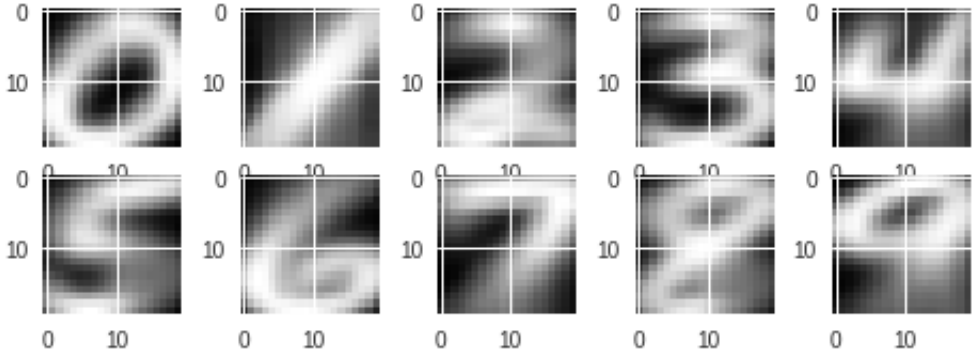


png

```
bernoulli_nb_stretched = BernoulliNaiveBayes()
bernoulli_nb_stretched.fit(features_train_stretched_boundingbox, labels_train)

print("BernoulliNaiveBayes distribution of stretched images")
for i in range(10):
    plt.subplot(4, 5, i+1)
    imshow(nb_stretched.means[i].reshape(20,20),cmap='gray')
```

```
BernoulliNaiveBayes distribution of stretched images
```



png

Part 2B: MNIST using Decision Forest

Classify MNIST using a decision forest.
For your forest construction, you should investigate four cases. Your cases are: number of trees = (10, 30) X maximum depth = (4, 16). You should compute your accuracy for each of the following cases: untouched raw pixels; stretched bounding box. This yields a total of 8 slightly different classifiers. Please use 20 x 20 for your bounding box dimensions.

```
print("10 trees + 4 depth + untouched: train & test accuracy- ", calculate_accuracy(False, 'decision_forest', True, (10,4)))

print("10 trees + 4 depth + stretched: train & test accuracy- ", calculate_accuracy(True, 'decision_forest', True, (10,4)))

print("10 trees + 16 depth + untouched: train & test accuracy- ", calculate_accuracy(False, 'decision_forest', True, (10,16)))

print("10 trees + 16 depth + stretched: train & test accuracy- ", calculate_accuracy(True, 'decision_forest', True, (10,16)))

print("30 trees + 4 depth + untouched: train & test accuracy- ", calculate_accuracy(False, 'decision_forest', True, (30,4)))

print("30 trees + 4 depth + stretched: train & test accuracy- ", calculate_accuracy(True, 'decision_forest', True, (30,4)))

print("30 trees + 16 depth + untouched: train & test accuracy- ", calculate_accuracy(False, 'decision_forest', True, (30,16)))

print("30 trees + 16 depth + stretched: train & test accuracy- ", calculate_accuracy(True, 'decision_forest', True, (30,16)))
```

```
10 trees + 4 depth + untouched: train & test accuracy-  (0.71845, 0.726)
10 trees + 4 depth + stretched: train & test accuracy-  (0.7444666666666667, 0.7663)
10 trees + 16 depth + untouched: train & test accuracy-  (0.99265, 0.9486)
10 trees + 16 depth + stretched: train & test accuracy-  (0.9958333333333333, 0.9549)

30 trees + 4 depth + untouched: train & test accuracy-  (0.76895, 0.7792)
30 trees + 4 depth + stretched: train & test accuracy-  (0.7681666666666667, 0.7867)
30 trees + 16 depth + untouched: train & test accuracy-  (0.9961833333333333, 0.9636)

30 trees + 16 depth + stretched: train & test accuracy-  (0.9974, 0.9646)
```

x	Method	Training Set Accuracy	Test Set Accuracy
1	Gaussian + untouched	0.54625	0.5415
2	Gaussian + stretched	0.8130333333333334	0.8234
3	Bernoulli + untouched	0.83645	0.8445
4	Bernoulli + stretched	0.795	0.8106
5	10 trees + 4 depth + untouched	0.71845	0.726
6	10 trees + 4 depth + stretched	0.7444666666666667	0.7663
7	10 trees + 16 depth + untouched	0.99265	0.9486
8	10 trees + 16 depth + stretched	0.9958333333333333	0.9549
9	30 trees + 4 depth + untouched	0.76895	0.7792
10	30 trees + 4 depth + stretched	0.7681666666666667	0.7867
11	30 trees + 16 depth + untouched	0.9961833333333333	0.9636
12	30 trees + 16 depth + stretched	0.9974	0.9646

References

Following are various resources referred while writing this solution

- My Github projects https://github.com/sriharshams/mlnd/
- Code of codyznash https://github.com/codyznash/GANs_for_Credit_Card_Data
- Tutorials of https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
- Naive Bayes in Scikit-Learn: Implementation of naive bayes in the scikit-learn library.
- Naive Bayes documentation: Scikit-Learn documentation and sample code for Naive Bayes
- Classify MNIST with PyBrain
- Scikit RandomForestClassifier
- Applied Machine Learning, D.A. Forsyth, (approximate 18'th draft)
- Piazza & Slack discussions on CS–498 Spring 2019